# LAB QUESTIONS

Q1. Design a class Matrix of dimension 3x3. Overload + operator to find sum of two matrices.

#include<iostream>

using namespace std;

class Matrix {

private:

int mat[3][3];

public:

Matrix() {

for(int i=0;i<3;i++)

for(int j=0;j<3;j++)

mat[i][j]=0;

}

void input() {

cout<<"Enter 3x3 matrix elements:\n";

for(int i=0;i<3;i++)

for(int j=0;j<3;j++)

cin>>mat[i][j];

}

void display() const {

for(int i=0;i<3;i++) {

for(int j=0;j<3;j++)

cout<<mat[i][j]<<" ";

cout<<endl;

}}

Matrix operator+(const Matrix& m) const {

Matrix result;

for(int i=0;i<3;i++)

for(int j=0;j<3;j++)

result.mat[i][j]=mat[i][j]+m.mat[i][j];

return result;

}};

int main() {

Matrix m1,m2,m3;

cout<<"For Matrix 1:\n";

m1.input();

cout<<"For Matrix 2:\n";

m2.input();

cout<<"Matrix 1:\n";

m1.display();

cout<<"Matrix 2:\n";

m2.display();

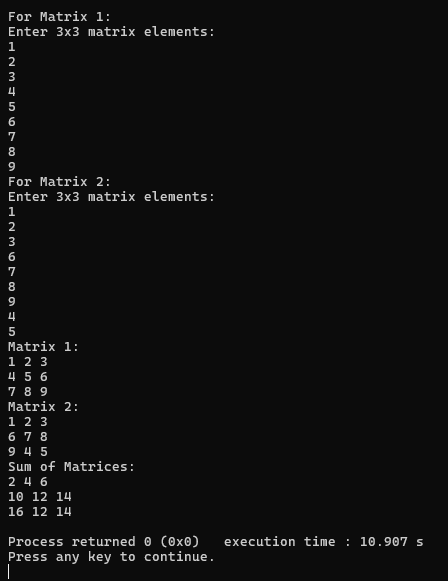
m3=m1+m2;

cout<<"Sum of Matrices:\n";

m3.display();

return 0;

}



QN2. Define a class string and use + and > operators to concatenate and compare two strings respectively.

#include<iostream>

#include<cstring>

using namespace std;

class String {

private:

char str[100];

public:

String() { str[0]='\0'; }

String(const char\* s) { strcpy(str,s); }

void input() {

cout<<"Enter a string: ";

cin>>str;

}

void display() { cout<<str<<endl; }

String operator+(const String& s) {

String result;

strcpy(result.str,str);

strcat(result.str,s.str);

return result;

}

bool operator>(const String& s) {

return strcmp(str,s.str)>0;

}};

int main() {

String s1,s2,s3;

cout<<"Input for First String:\n";

s1.input();

cout<<"Input for Second String:\n";

s2.input();

cout<<"First String: ";

s1.display();

cout<<"Second String: ";

s2.display();

s3=s1+s2;

cout<<"Result after Concatenation: ";

s3.display();

if(s1>s2)

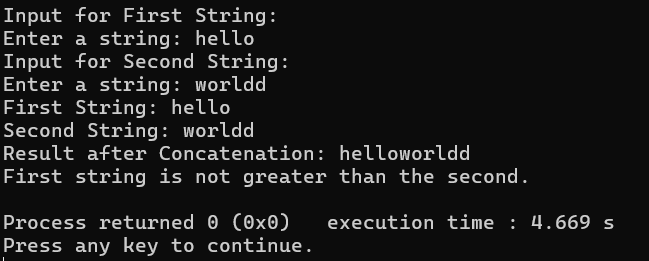
cout<<"First string is greater than the second."<<endl;

else

cout<<"First string is not greater than the second."<<endl;

return 0;

}



Q3. Write a program to implement vector addition and subtraction using operator overloading.

#include<iostream>

using namespace std;

class Vector {

private:

double x,y,z;

public:

Vector(double x=0,double y=0,double z=0):x(x),y(y),z(z){}

void input() {

cout<<"Enter vector components (x y z): ";

cin>>x>>y>>z;

}

void display() {

cout<<"("<<x<<", "<<y<<", "<<z<<")"<<endl;

}

Vector operator+(const Vector& v) {

return Vector(x+v.x,y+v.y,z+v.z);

}

Vector operator-(const Vector& v) {

return Vector(x-v.x,y-v.y,z-v.z);

}};

int main() {

Vector v1,v2,v3,v4;

cout<<"Enter details for Vector 1:\n";

v1.input();

cout<<"Enter details for Vector 2:\n";

v2.input();

cout<<"Vector 1: ";

v1.display();

cout<<"Vector 2: ";

v2.display();

v3=v1+v2;

v4=v1-v2;

cout<<"Vector Addition: ";

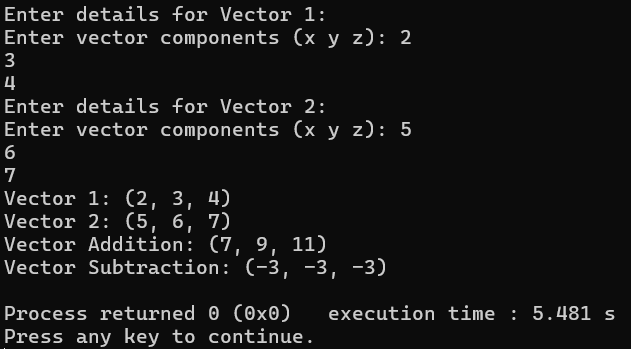
v3.display();

cout<<"Vector Subtraction: ";

v4.display();

return 0;

}



Q4. Design a class Matrix, overload ++ and -- operator to increment and decrement each element of the matrix by 1.

#include <iostream>

using namespace std;

class Matrix {

private:

int mat[3][3];

public:

Matrix() {

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

mat[i][j] = 0;

}

void input() {

cout << "Enter 3x3 matrix elements:\n";

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

cin >> mat[i][j];

}

void display() {

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++)

cout << mat[i][j] << " ";

cout << endl;

}

}

Matrix& operator++() {

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

mat[i][j]++;

return \*this;

}

Matrix& operator--() {

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

mat[i][j]--;

return \*this;

}

Matrix operator++(int) {

Matrix temp = \*this;

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

mat[i][j]++;

return temp;

}

Matrix operator--(int) {

Matrix temp = \*this;

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

mat[i][j]--;

return temp;

}

};

int main() {

Matrix m;

cout << "Enter elements:\n";

m.input();

cout << "Original:\n";

m.display();

cout << " Prefix ++:\n";

++m;

m.display();

cout << " Postfix ++:\n";

m++;

m.display();

cout << " Prefix --:\n";

--m;

m.display();

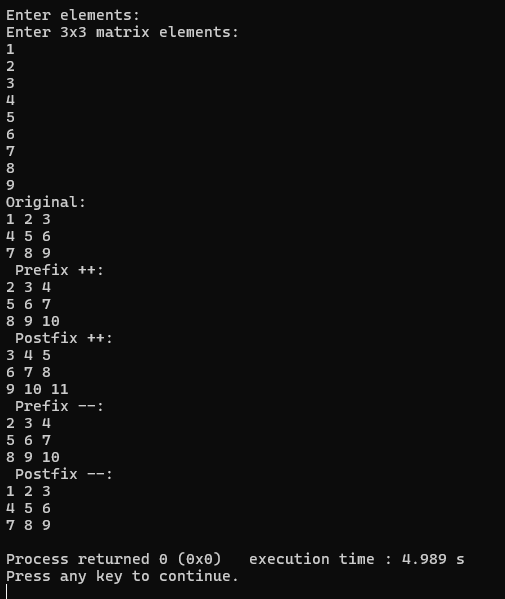
cout << " Postfix --:\n";

m--;

m.display();

return 0;

}



Q5. Write a program to access elements of a vector class with index operator.

#include <iostream>

using namespace std;

class Vector {

private:

int arr[5];

int size;

public:

Vector() : size(5) {

for (int i = 0; i < size; i++)

arr[i] = 0;

}

void input() {

cout << "Enter " << size << " elements: ";

for (int i = 0; i < size; i++)

cin >> arr[i];

}

void display() {

for (int i = 0; i < size; i++)

cout << arr[i] << " ";

cout << endl;

}

int& operator[](int index) {

if (index >= 0 && index < size)

return arr[index];

cout << "error" << endl;

return arr[0];

}};

int main() {

Vector v;

cout << "Input elements:\n";

v.input();

cout << "Original: ";

v.display();

cout << "Accessing element at index 3: " << v[2] << endl;

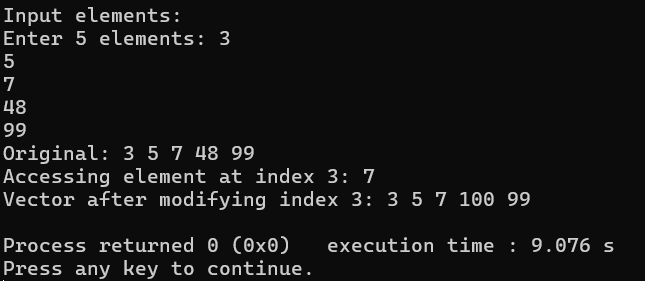
v[3] = 100;

cout << "Vector after modifying index 3: ";

v.display();

return 0;

}



Q6. Write a program to multiply two matrices by overloading the + operator.

#include <iostream>

using namespace std;

class Matrix {

private:

int mat[3][3];

public:

Matrix() {

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

mat[i][j] = 0;

}

void input() {

cout << "Enter 3x3 matrix elements:\n";

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

cin >> mat[i][j];

}

void display() {

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++)

cout << mat[i][j] << " ";

cout << endl;

}

}

Matrix operator\*(const Matrix& m) {

Matrix result;

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

result.mat[i][j] = mat[i][j] \* m.mat[i][j];

return result;

}

};

int main() {

Matrix m1, m2, m3;

cout << "For Matrix 1:\n";

m1.input();

cout << "For Matrix 2:\n";

m2.input();

cout << "Matrix 1:\n";

m1.display();

cout << "Matrix 2:\n";

m2.display();

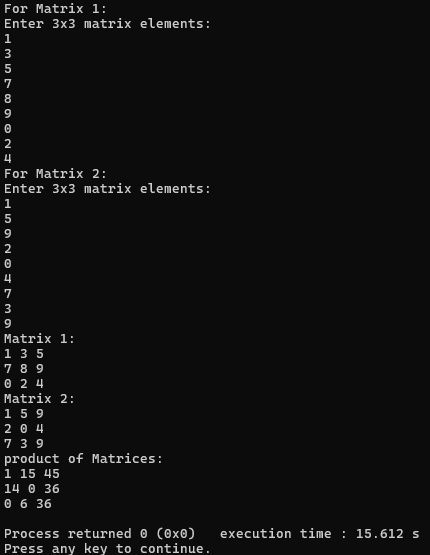
m3 = m1 \* m2;

cout << "product of Matrices:\n";

m3.display();

return 0;

}



Q7. Create a class named City that will have two member variables CityName and DistFromKtm (float). Add member functions to set and retrieve the CityName and DistFromKtm separately. Add operator overloading to find the distance between the cities (just find the difference of DistFromKtm) and sum of distance of those cities from Kathmandu. In the main function, initialize three city objects. Set the first and second city to be Pokhara and Dhangadi. Display the sum of DistFromKtm of Pokhara and Dhangadi and distance between Pokhara and Dhangadi..

#include <iostream>

#include <string>

using namespace std;

class City {

private:

string CityName;

float DistFromKtm;

public:

void setCityName(string name) {

CityName = name;

}

void setDistFromKtm(float dist) {

DistFromKtm = dist;

}

string getCityName() {

return CityName;

}

float getDistFromKtm() {

return DistFromKtm;

}

float operator+(City c) {

return DistFromKtm + c.DistFromKtm;

}

float operator-(City c) {

return DistFromKtm - c.DistFromKtm;

}

};

int main() {

City c1, c2, c3;

c1.setCityName("Pokhara");

c1.setDistFromKtm(125);

c2.setCityName("Dhangadi");

c2.setDistFromKtm(236);

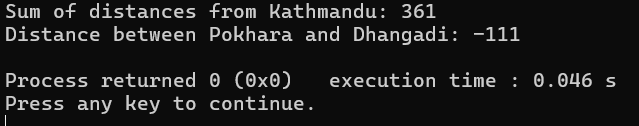
cout << "Sum of distances from Kathmandu: " << c1 + c2 << endl;

cout << "Distance between " << c1.getCityName() << " and " << c2.getCityName()

<< ": " << c1 - c2 << endl;

return 0;

}



Q8. Write a program to overload the relational operators to compare the length (in meter and centimeter) of two objects.

#include <iostream>

using namespace std;

class Distance {

private:

float m;

float cm;

public:

Distance() {

m = 0.0f;

cm = 0.0f;

}

Distance(float meters, float centimeters) {

this->m = meters;

this->cm = centimeters;

while (cm >= 100.0f) {

cm -= 100.0f;

m += 1.0f;

}}

void getDistance() {

cout << "Enter meter: ";

cin >> m;

cout << "Enter centimeter: ";

cin >> cm;

}

void showDistance() {

cout << m << "m-" << cm << "cm";

}

friend bool operator>(Distance& d1, Distance& d2);

friend bool operator==(Distance& d1, Distance& d2);

};

bool operator>(Distance& d1, Distance& d2) {

float totalCm1 = d1.m \* 100.0f + d1.cm;

float totalCm2 = d2.m \* 100.0f + d2.cm;

return totalCm1 > totalCm2;

}

bool operator==(Distance& d1, Distance& d2) {

float totalCm1 = d1.m \* 100.0f + d1.cm;

float totalCm2 = d2.m \* 100.0f + d2.cm;

return totalCm1 == totalCm2;

}

int main() {

Distance dist1(5.0f, 29.3f);

Distance dist2(2.0f, 56.9f);

cout << "Distance 1: ";

dist1.showDistance();

cout << "Distance 2: ";

dist2.showDistance();

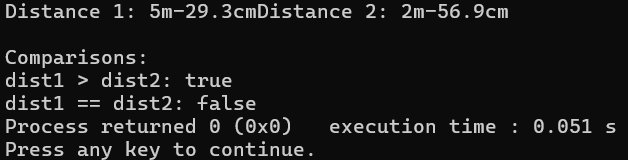
cout << endl << endl << "Comparisons:";

cout << endl << "dist1 > dist2: " << ((dist1 > dist2) ? "true" : "false");

cout << endl << "dist1 == dist2: " << ((dist1 == dist2) ? "true" : "false");

return 0;

}



Q9. Write operator functions as member function of a class to overload arithmetic operator +, logical operator <=, and stream operator << to operate on the objects of user-defined type time (hr, min, sec).

#include <iostream>

#include <iomanip>

using namespace std;

class Time {

private:

int days;

int hours;

int minutes;

int seconds;

public:

Time() {

days = 0;

hours = 0;

minutes = 0;

seconds = 0;

}

void getTime() {

cout << "Enter hours: ";

cin >> hours;

cout << "Enter minutes: ";

cin >> minutes;

cout << "Enter seconds: ";

cin >> seconds;

normalize();

}

void normalize() {

minutes += seconds / 60;

seconds = seconds % 60;

hours += minutes / 60;

minutes = minutes % 60;

days += hours / 24;

hours = hours % 24;

}

friend ostream& operator<<(ostream& os, const Time& t) {

if (t.days > 0) {

os << t.days << " day";

if (t.days > 1) os << "s";

os << ", ";

}

os << t.hours << ":"

<< setw(2) << setfill('0') << t.minutes << ":"

<< setw(2) << setfill('0') << t.seconds;

return os;

}

Time operator+(Time& t2) {

Time result;

result.seconds = seconds + t2.seconds;

result.minutes = minutes + t2.minutes;

result.hours = hours + t2.hours;

result.days = days + t2.days;

result.normalize();

return result;

}

bool operator>=(Time& t2) {

int totalSeconds1 = days \* 24 \* 60 \* 60 + hours \* 60 \* 60 + minutes \* 60 + seconds;

int totalSeconds2 = t2.days \* 24 \* 60 \* 60 + t2.hours \* 60 \* 60 + t2.minutes \* 60 + t2.seconds;

return totalSeconds1 >= totalSeconds2;

}

bool operator<=(Time& t2) {

int totalSeconds1 = days \* 24 \* 60 \* 60 + hours \* 60 \* 60 + minutes \* 60 + seconds;

int totalSeconds2 = t2.days \* 24 \* 60 \* 60 + t2.hours \* 60 \* 60 + t2.minutes \* 60 + t2.seconds;

return totalSeconds1 <= totalSeconds2;

}

};

int main() {

Time t1, t2, sum;

cout << "Enter first time:" << endl;

t1.getTime();

cout << "\nEnter second time:" << endl;

t2.getTime();

sum = t1 + t2;

cout << "\nTime 1: ";

cout << t1;

cout << "\nTime 2: ";

cout << t2;

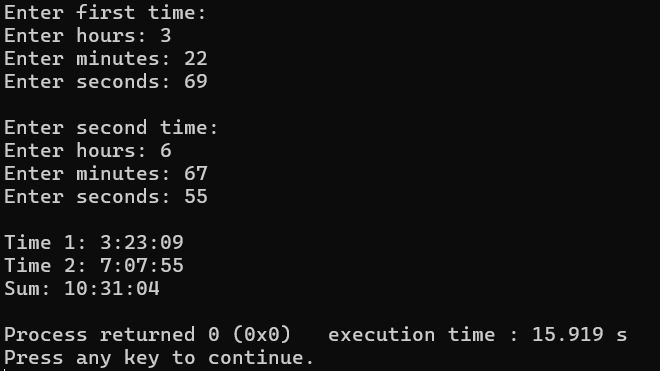
cout << "\nSum: ";

cout << sum;

cout<<endl;

return 0;

}



Q10. Create a class called time that has separate int member data for hours, minutes, and seconds. One constructor should initialize this data to zero (0), and another should initialize it to fixed values. A member function should display it in 10:45:30 format. The final member function should add two objects of type time passed as arguments using operator overloading.

#include <iostream>

using namespace std;

class Time {

private:

int days;

int hours;

int minutes;

int seconds;

public:

Time() {

days = 0;

hours = 0;

minutes = 0;

seconds = 0;

}

void getTime() {

cout << "Enter hours: ";

cin >> hours;

cout << "Enter minutes: ";

cin >> minutes;

cout << "Enter seconds: ";

cin >> seconds;

normalize();

}

void normalize() {

minutes += seconds / 60;

seconds = seconds % 60;

hours += minutes / 60;

minutes = minutes % 60;

days += hours / 24;

hours = hours % 24;

}

void showTime() {

if (days > 0) {

cout << days << " day";

if (days > 1) cout << "s";

cout << ", ";

}

cout << hours << ":" << (minutes < 10 ? "0" : "") << minutes << ":" << (seconds < 10 ? "0" : "") << seconds;

}

Time operator+(Time& t2) {

Time result;

result.seconds = seconds + t2.seconds;

result.minutes = minutes + t2.minutes;

result.hours = hours + t2.hours; // fixed here

result.days = days + t2.days;

result.normalize();

return result;

}

};

int main() {

Time t1, t2, sum;

cout << "Enter first time:" << endl;

t1.getTime();

cout << "\nEnter second time:" << endl;

t2.getTime();

sum = t1 + t2;

cout << "\nTime 1: ";

t1.showTime(); // use showTime to display

cout << "\nTime 2: ";

t2.showTime();

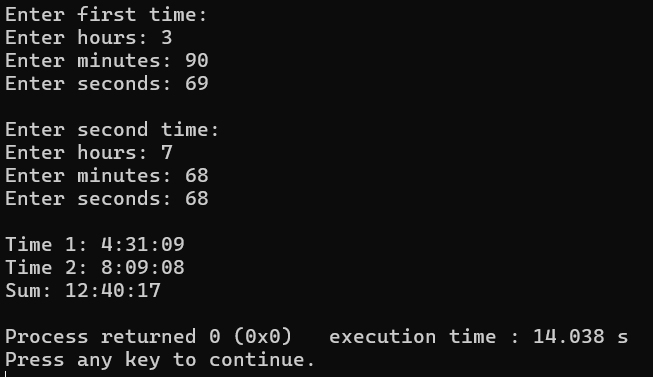
cout << "\nSum: ";

sum.showTime();

cout << endl;

return 0;

}



# DISCUSSIONS

In this lab session, we discussed about the operator overloading. Operator overloading allows us to redefine the meaning of operators for user-defined types such as classes. We learnt about the operators that can be overloaded and some operators which cant be overloaded such as member access operator (.), pointer-to-member access operator (.\*), scope resolution operator (::), and the ternary conditional operator and type cast operators like const\_cast, static\_cast, reinterpret\_cast, and dynamic\_cast. We observed how operator overloading provides a clean and intuitive interface to work with class objects following object-oriented programming principles.

# CONCLUSIONS

In conclusion, this lab session on operator overloading helped us understand how operator overloading helps us in maintaining the code reusability and readability. This feature supports the principles of object-oriented programming, such as abstraction and encapsulation, by allowing clean interfaces and hiding complex implementation details.